Using Process Simulation to Compare Scheduling Strategies for Software Projects

Frank Padberg
Fakultät für Informatik
Universität Karlsruhe, Germany
padberg@ira.uka.de

Abstract

We present a discrete simulation model for software projects which explicitly takes a scheduling strategy as input. The model represents varying staff skill levels, component coupling, rework caused by design changes, and changing task assignments. The simulation model is implemented in the ModL language of the general-purpose graphical simulation tool EXTEND. The simulations provide quick feedback about the impact which the scheduling strategy will have on the progress and completion time of a given software project. Using the model, a manager can compare different strategies and choose the one which is best for his next project.

As an illustration how to apply the simulation model, we systematically study the performance of various list policies for a small sample project. We provide a detailed analysis of the task assignments which actually occur in the simulations. In addition, the example provides clear evidence that strategies which are more adaptive to the current project state than list policies will yield improved schedules. This result suggests to apply dynamic optimization techniques when scheduling software projects.

1. Introduction

To cut development cost and meet tight deadlines in short staffed software projects, it is essential that managers plan and schedule their projects best possible. Good software project scheduling is a hard task in practice. The time needed to complete a software development activity is difficult to estimate since it depends not only on technical factors, but also on human factors such as the experience of the developers. Even worse, it is typical for software projects that the completion of tasks is delayed because of unanticipated rework which was caused by feedback in the development process.

In this paper, we show how to use *process simulation* to evaluate scheduling strategies for software projects. We present a discrete-time stochastic simulation model which is tailored to software projects and explicitly takes a scheduling strategy (or *policy*) as input. A strategy specifies for each possible state of the project which action to take, such as reassigning some task. In each simulation run, one possible full path of the project is simulated. The output of a series of simulation runs can be used to measure the performance of a given policy. For example, the output of the simulations for the fixed policy can be combined into a probability distribution for the project completion time.

The simulation model is an *implementation* of the probabilistic scheduling model for software projects which we have presented earlier [7]. In the scheduling model, teams work in parallel on the software's components. Unplanned changes in the system design can occur at any time and lead to rework. Since the components are coupled, for example, through common interfaces, changes which originate in one part of the system can propagate to other parts of the system. As input to the model, statistical data collected during past projects and high-level design data about the current project are required, see subsection 2.5. By modelling individual components, our model is more fine-grained than system dynamics models [1, 4, 9]. The way in which our model describes feedback between the activities is novel in scheduling [5, 6, 8].

The simulation model facilitates experimentation. For example, a manager can see quickly how the completion time of his project changes when he changes some of the input parameters. In particular, the simulations provide quick feedback about the impact which a particular scheduling strategy will have on the progress of the project. This way, a manager can evaluate and compare different strategies and choose the one which he thinks is best for his next project.

As an application, we use simulation to systematically study all possible *list policies* for a small sample project. A list policy prescribes an order in which the components must be worked on, but the actual assignment of the components to the teams depends on the progress of the project: the next team to finish must work on the next component in the list. List policies are a commonly used class of scheduling strategies. The sample project consists of four components and two teams. Although the project is small, it is not clear aforehand which list policy the manager should prefer because of the probabilistic nature of the process. Based on a detailed analysis of the performance of the task assignments which actually occur in the simulations, we are able to identify three completely different ways to achieve a good average completion time for the sample project. In addition, we find that the performance of a particular task assignment is sensitive to the project context in which it occurs. These results provide clear evidence that scheduling strategies which are more adaptive to the current project state than list policies will on average yield much better schedules for software projects.

2. Simulation Model

2.1. Project dynamics

The simulation model is an implementation of the stochastic scheduling model for software projects which we have presented earlier [7]. The model captures much of the dynamics of software projects, representing varying staff skill levels, design changes, component coupling, rework, and changing task assignments.

The software product is developed by several *teams.* Based on some early high-level design, the software is divided into *components.* At any time during the project, each team works on at most one component, and, vice versa, each component is being worked on by at most one team. The assignment of the components to the teams may change during the project. It is not required that a team has completed its current component before it is allocated to some other component; a team may be *interrupted* and *re-allocated* to another component by the manager.

The teams do not work independently. From time to time a team might detect a problem with the software's high-level design. Since the components are coupled, for example, through common interfaces, such a problem is likely to affect other components and teams as well. To eliminate the problem, the high-level design gets revised. Some of the components will have to be *reworked* because of the design changes while other components are not affected. This way, the progress that a team makes developing its component depends on the progress of the other teams (feedback).

2.2. Scheduling actions

In the model, a project advances through a sequence of *phases.* By definition, a phase ends when staff becomes available or when the software's high-level design must be changed. Staff becomes available when some team completes its component. Staff also becomes available when some team completes all rework on a component which already had been completed earlier in the project but had to be reworked because of a design change. Each phase lasts for some number of discrete *time slices.*

Scheduling actions take place only at the end of a phase. Possible scheduling actions are: assigning a component to a team, starting a team, and stopping a team. Scheduling at arbitrary points in (discrete) time is not modelled. The rationale behind this restriction is that is does not make sense to re-schedule a project as long as nothing unusual happens. At the end of a phase though, staff is available again for allocation, or re-scheduling the project might be appropriate because of some design changes. At that time, the manager may also interrupt some of the teams and re-allocate them to other components.

2.3. Project state

The *state* of a project changes at the end of each phase. The state of a project by definition includes: a progress vector, a rework vector, and a countdown.

The *progress vector* has one entry for each component. The progress of a component is defined as the *net* development time that has been spent working on the component. The net development time is obtained from the total development time by substracting all rework times spent for adapting the component to high-level design changes.

The *rework vector* has one entry for each component, too. The rework time for a component is the time that yet must be spent with adapting the component to high-level design changes. As soon as a component's rework time has been counted down to zero, "normal" development of the component can proceed. Once a component has been completed, only rework may occur for the component in the sequel.

The *countdown* is the time left until the project's *deadline* will be reached. If the deadline is exceeded, the project will be cancelled as a failure. For more details on the definitions, please refer to [7].

2.4. Probabilities

The scheduling model is probabilistic. That is, events will occur only with a certain probability at a particular point in time. In particular, the point in time at which some component is completed, the points in time at which design changes occur, the set of components which are affected by a design change, and the amount of rework caused by a design change are subject to chance.

Given a project state ζ and a scheduling action a, the state of the project at the end of the next phase will be equal to η only with a certain probability, called the *transition probability* $\mathrm{P}(\zeta, a; \eta)$. The transition probability does not depend on any information about the project's history except its current state and the scheduling action chosen. The resulting process

$$\zeta(0), \ a(0), \ \zeta(1), \ a(1), \ \ldots$$

is a Markov decision process [2]. To compute the transition probabilities, statistical data about past projects and high-level design data are required as input, see the next subsection.

2.5. Input data

The input data for the simulation model are:

- the base probabilities,
- the probabilities of rework times,
- the dependency degrees,
- the scheduling strategy.

The scheduling strategy specifies for each possible state of the project which scheduling action to take. The other input data are required to compute the transition probabilities in the scheduling model, respectively, determine the next step in a simulation run (see the next subsection).

The *base probabilities* are a measure for the pace at which the teams have made progress in previous projects. For each team and component, there is a separate set of base probabilities which specify how likely it is that the team will finish the component, or report a high-level design problem, after a specific amount of time. The base probabilities depend upon various human and technical factors, for example, the software process employed by the team, the complexity of the component to be developed, and the skills of the team. The base probabilities are computed from empirical data collected during past projects and reflect the specific development environment in a company. In addition, for each component there is a probability distribution which specifies the amount of *rework time* required if the component has to be reworked because of a design change.

The *dependency degrees* are a probabilistic measure for the strength of the coupling between the components. The stronger the coupling is the more likely it is that high-level design problems which originate in one component will propagate to other components, leading to rework. The dependency degree $\alpha(K, X)$ by definition is the probability that changes in the software's design will extend over exactly the set X of components given that the problems causing the redesign were detected in the set K of components. The dependency degrees are computed from the high-level design of the software. Thus, the model explicitly takes the design of the software as input.

2.6. Implementation

The simulation model is a discrete-time simulation written in the ModL language of the general-purpose graphical simulation tool EXTEND [3]. The scheduling strategy is implemented as a separate block in the simulation and thus can be easily exchanged.

For a given scheduling strategy and project state, the simulation model determines which step the project will take next (that is, whether some component will be finished in the next step or whether a design change will occur) by "throwing a dice". The dice behaves according to the base probability distributions, taking into account the current state of the project.

Similarly, if a design change occurs in the project the simulation determines the set of components which are affected by the design change by throwing another dice which behaves according to the dependency degrees. Afterwards, the amount of rework time required for each of the affected components is determined using yet another dice which behaves according to the probabilities of rework time. This way, in each simulation run one possible full path of the project is simulated.

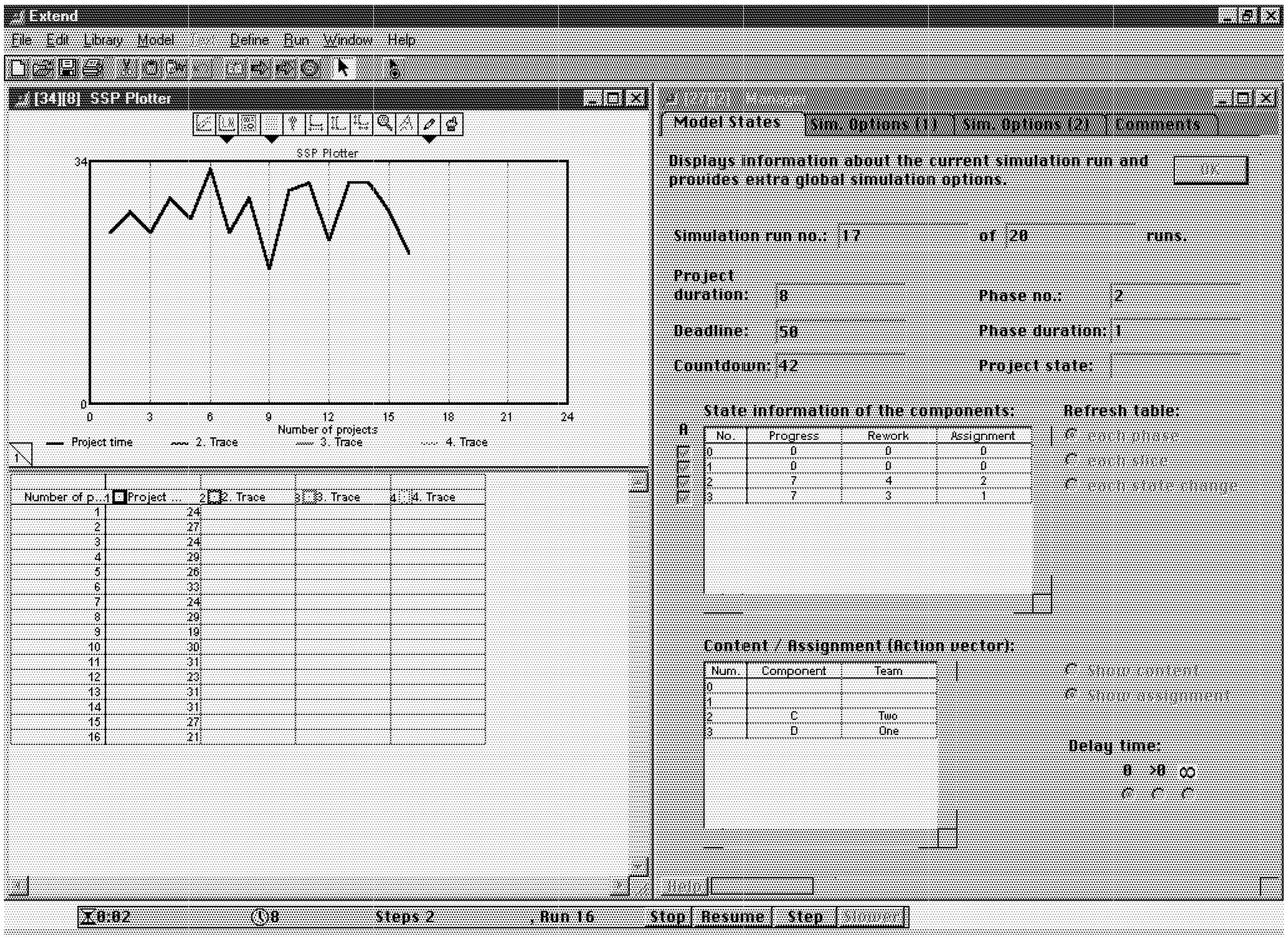
Figure 1 is a screenshot showing a simulation in progress. The window on the right half of the screen shows the block of the simulation model where the simulation setup and the current state of the project are displayed. The state of the project includes the net development time spent on each component so far, the project duration up to this point, the amount of rework left for each component, and the current task assignment. The window on the left half of the screen shows a plotter which observes the project completion time for all simulation runs.

3. Sample Project

3.1. Architecture

The sample project may be thought of as a client-server system which consists of four components. The client contains a front-end, component A, and a large application part, component C, which does some pre-processing. The server contains an administrator front-end, component B, and a large application kernel, component D. The client and the

Figure 1. A simulation in progress.



Number of p...	1 Project ...	2 2. Trace	3 3. Trace	4 4. Trace
1	24			
2	27			
3	24			
4	29			
5	26			
6	33			
7	24			
8	29			
9	19			
10	30			
11	31			
12	23			
13	31			
14	31			
15	27			
16	21			

Displays information about the current simulation run and provides extra global simulation options.

Simulation run no.: 17 of 20 runs.

Project duration: 8 Phase no.: 2

Deadline: 50 Phase duration: 1

Countdown: 42 Project state:

State information of the components:

No.	Progress	Rework	Assignment
0	0	0	0
1	0	0	0
2	7	4	2
3	7	3	1

Refresh table: [✓ each phase] each slice each state change

Content / Assignment (Action vector):

Num.	Component	Team
0		
1		
2	C	Two
3	D	One

Show content [✓ Show assignment]

Delay time: [✓ 0] >0 ∞

0:02 8 Steps 2 , Run 16

server are coupled only through the application components C and D. In particular, there is no direct link between the two front-end components.

3.2. Base probabilities

There are two teams in the project, team One and team Two. The complexity of the components and the productivity of the teams are reflected in the probability distributions which are used as input for the simulations. For each team and component, we must specify two base distributions: one distribution describes the net development time, the other distribution describes the problem reporting time. We assume that a time slice corresponds to one week.

For the sample project, the distributions were generated from binomial distributions with suitable length (number of non-zero bars) and skewness. The parameters chosen for the binomial distributions are listed in Table 1. The distributions were rounded to one decimal digit and scaled such that for a fixed team and component the probabilities of the net development times and problem reporting times sum up to 100 percent. From the x-values where the distributions assume their peak value one can see that *on average*

- team Two has a 30 percent lower productivity than team One,
- the net development times for the front-end components A and B are about the same for a given team,
- the net development times for the core application components C and D are much longer than for the front-end components,
- the risk that high-level design changes will occur is much higher for the application components than for the front-end components,
- design changes can be expected to occur mainly after two thirds of the net development time of a component,

Table 1. Base probabilities.

comp	team	distrib	length	skew	peak	scale
A	One	net dev	8	0.7	6	80 %
		report	7	0.5	4	20 %
A	Two	net dev	11	0.7	8	80 %
		report	10	0.5	6	20 %
B	One	net dev	8	0.8	7	80 %
		report	7	0.55	4	20 %
B	Two	net dev	11	0.8	9	80 %
		report	10	0.55	7	20 %
C	One	net dev	13	0.75	10	60 %
		report	12	0.45	7	40 %
C	Two	net dev	17	0.75	13	60 %
		report	16	0.45	10	40 %
D	One	net dev	15	0.75	12	40 %
		report	14	0.45	8	60 %
D	Two	net dev	21	0.75	16	40 %
		report	20	0.45	13	60 %

Table 2. Rework time probabilities.

comp	length	skew	peak
A	2	0.25	1
B	2	0.25	1
C	5	0.5	3
D	5	0.55	3

- component D has the largest expected effort and is a high risk component.

In addition to the base probabilities, we must specify for each component a distribution which describes the amount of rework time required if that component is affected by a design change. Again, we use binomial distributions, see Table 2.

3.3. Dependency degrees

The strength of the coupling between the components is reflected by the dependency degrees. For the sample project, the values are chosen in such a way that

- the core components C and D are strongly coupled,
- each front-end is strongly coupled to its application component,
- changes must propagate along the interfaces between the components,
- there is only a limited risk that a design change which originates in a front-end, say component A, will propagate to the *other* application component, in this case component D,
- design changes in one front-end can have an impact on the other front-end only if the intermediate components C and D are affected.

The dependency degrees are listed in Table 3. Blank entries correspond to zero.

3.4. List policies

A list policy uses a fixed priority list for the components to prescribe an order in which the components must be developed. When a team finishes its current task, it is allocated to the next unprocessed component in the list. Since we have four components, there are $\text{fac}(4) = 24$ different list policies for the sample project.

In a probabilistic setting, the task completion times are not known in advance. Thus, the priority list does not completely pre-determine to which team a particular component will actually get assigned. The actual schedule (task assignments and their timing) depends on the order in which the teams finish their tasks, which is subject to chance. An exception are the components which are assigned right at the beginning of the project. For example, the list policy CDAB will initially assign component C to team One and component D to team Two. Whichever team finishes its task first will work on component A. Finally, the next team to finish will work on component B.

Usually, a team works on its component until completion without interruption. An exception is when the team has to rework one of its previously completed components because of a design change. After having finished the rework, the team resumes working on the uncompleted component.

3.5. Simulation results

Even for such a small sample project it is *not* obvious which list policy a manager should prefer because of the probabilistic nature of the development process and the feedback between activities. To find the best list policy for the sample project, we run 500 project simulations for each of the 24 possible lists. For each list, we then compute a histogram for the project completion time from the 500 observed project completion times. Table 4 summarizes the observed performance of each list policy.

For example, the mean development time (fourth column) for list policy BCDA is about 16 percent shorter than for list policy CBAD. Even more important to a manager in practice is the fact that with BCDA the project has a 75 percent chance (second column) to finish after 28 weeks, whereas with CBAD the manager must plan for 34 weeks (or 21 percent more time) to reach the 75 percent threshold. A similar result holds for a threshold of 90 percent (third column).

Table 3. Dependency degrees.

	A	B	C	D	AB	AC	AD	BC	BD	CD	ABC	ABD	ACD	BCD	ABCD
A	20					60							15		5
B		20							60					15	5
C			10			15				35			20	10	10
D				10					15	35			10	20	10
AB					20						20	20			40
AC						30							55		15
AD							10					10	50		30
BC								10			10			50	30
BD									30					55	15
CD										40			20	20	20
ABC											10				90
ABD												10			90
ACD													40		60
BCD														40	60
ABCD															100

Table 4. Performance of the list policies.

list policy	75 %	90 %	mean
ABCD	32	37	29.8
ABDC	29	33	27.1
ACBD	33	37	30.4
ACDB	28	31	26.2
ADBC	30	35	28.0
ADCB	29	32	26.8
BACD	31	36	29.2
BADC	29	33	27.2
BCAD	33	37	30.7
BCDA	28	31	25.7
BDAC	30	34	27.9
BDCA	29	33	26.9
CABD	34	40	30.7
CADB	32	36	29.3
CBAD	34	40	30.9
CBDA	32	37	29.3
CDAB	30	33	27.4
CDBA	29	33	27.4
DABC	33	38	29.7
DACB	28	32	26.0
DBAC	32	37	29.3
DBCA	29	33	26.8
DCAB	28	32	26.6
DCBA	28	32	26.2

On average, the project completion times are longer than might be expected when looking at the binomial distributions for the development times of the individual components. Recall that these distributions model just the *net* development times; the actual development times observed in the simulations are longer due to the rework caused by design changes.

We can make a number of interesting observations about the simulation results:

1. The list policies which assign the large components C and D first (CDAB, CDBA, DCAB, and DCBA) all perform well. Yet, some lists which assign the small front-end components first also perform well (ABDC and BADC).
2. Making sure that the faster team One starts with the largest component D does not automatically lead to a good schedule; the lists DABC and DBAC on average take much longer to complete than the other lists which assign component D first.
3. The lists where team One starts with component C all perform worse than their counterparts where team One starts with component D.
4. The lists where team Two, which has the lower productivity, starts with the largest component D rank in the middle of the field.
5. The lists which assign the largest component last all are a bad choice.
6. The list policies which assign the front-end components second and third (CABD, CBAD, DABC, and DBAC) all are a bad choice.

Again, these results refer to the *average* performance of the list policies. The results are analysed and explained in detail in the next section.

4. Analysis

4.1. Actual assignments

In order to understand why a particular list policy shows the performance observed in the simulations, the task assignments which *actually occur* during the simulated projects are of key importance. Therefore, we observe for each simulation run and each component which team was allocated to the component. To specify an assignment, we use a 4-digit notation. The first digit is the number of the team which was allocated to component A, the second digit is the number of the team which was allocated to component B, and so on. For example, to specify that team One was allocated to components B and D, while team Two was allocated to components A and C, we use the notation 2121.

Table 5 shows for each list policy the task assignments and the relative frequency with which they have occured among the 500 simulation runs for that list policy. Only those assignments which occured in more than 10 percent of the runs are listed. For each assignment, the table also shows the average performance of the assignment using the same measures as were used in the preceding section for the list policy as a whole.

For example, the list policy ADCB resulted in the task assignment 1112 in 57 percent of the 500 simulations for that policy, or 286 simulations. The mean development time was 25.6 weeks for these 286 projects, and there was a 75 percent chance to finish the project after 27 weeks. On the other hand, in 43 percent of the simulations policy ADCB resulted in the task assignment 1212. The mean development time for these projects was longer, namely, 28.4 weeks. The average performance of list policy ADCB is a mixture of the peformance for the two assignments 1112 and 1212.

4.2. Good and bad policies

Policy BCDA in almost all cases leads to the assignment 2121. With that assignment, each team works on one large application component and one small front-end component. Such an assignment is called a *balanced assignment.* In addition, the faster team One works on the more difficult application component D. This seems to be a favorable task assignment. Policy ACDB in most cases leads to the similar assignment 1221 (where just the front-end components are switched) and also shows a good performance. The same reasoning applies to the policies DACB, DCBA, DCAB, DBCA, ABDC, and BADC.

Table 5. Actual assignments and their performance.

policy	assign	freq	75 %	90 %	mean
ABCD	1212	0.95	32	37	29.9
ABDC	1221	0.96	29	33	27.1
ACBD	1121	0.78	33	38	30.2
	1122	0.22	33	35	31.3
ACDB	1221	0.91	28	31	26.2
ADBC	1112	0.97	30	35	28.0
ADCB	1112	0.57	27	31	25.6
	1212	0.43	31	34	28.4
BACD	2112	0.82	31	35	29.5
	2121	0.18	30	37	27.5
BADC	2121	0.84	29	33	27.2
	2112	0.16	30	33	27.2
BCAD	1121	0.77	32	37	30.0
	1122	0.23	34	39	32.7
BCDA	2121	0.95	28	31	25.7
BDAC	1112	0.97	30	34	27.9
BDCA	1112	0.56	28	32	26.6
	2112	0.44	29	33	27.3
CABD	2211	0.82	34	39	30.6
	2112	0.14	31	34	28.6
CADB	2112	0.85	33	37	29.8
	2211	0.15	28	33	26.1
CBAD	2211	0.68	34	40	30.3
	1212	0.28	31	39	30.7
CBDA	1212	0.74	33	38	30.3
	2211	0.26	28	34	26.2
CDAB	1112	0.57	29	33	26.5
	1212	0.41	30	33	28.5
CDBA	1112	0.50	29	33	26.8
	2112	0.48	30	33	27.9
DABC	2211	0.79	32	36	28.5
	2221	0.16	40	45	36.7
DACB	2121	0.93	28	32	26.1
DBAC	2211	0.80	31	35	28.0
	2221	0.15	39	44	36.7
DBCA	1221	0.93	29	33	27.1
DCAB	1221	0.59	28	32	26.6
	2121	0.32	28	31	26.3
DCBA	2121	0.62	28	32	25.9
	1221	0.31	29	33	27.0

A balanced assignment where the slower team Two works on the difficult component D in general is much less preferable. For example, policy BACD in the majority of the projects leads to the assignment 2112 and on average requires a much longer development time (29 weeks). The same holds for the policies CADB, CBDA, and ABCD.

A surprising alternative to a balanced assignment with component D assigned to the faster team One is revealed by the policy ADCB. In about half of the projects, policy ADCB leads to the assignment 1112. With that assignment, the slower team Two works on the difficult application component D, but all the remaining components are assigned to the other, fast team. Such an assignment yields a good performance for the policies ADCB, BDCA, CDBA, and CDAB. The reason why these policies do not rank as high as, for example, policy BCDA, is that they have a more than 40 percent chance of leading to one of the less favorable balanced assignments 1212 and 2112.

As opposed to policies such as ADCB, the policies ADBC and BDAC almost always lead to the assignment 1112, but do not exhibit as good a performance (28 weeks completion time on average). We shall discuss this observation in detail in the next subsection.

Policies which assign *both* large components to the same team in general are not a good choice. Policies DBAC, DABC, CABD, and CBAD in about three out of four projects lead to the assignment 2211. The average completion time in this case ranges between 28 and 31 weeks. Assigning the large application components to the slower team Two is even worse, of course; this assignment occurs in about 20 percent of the projects for the policies ACBD and BCAD, and results in an average project completion time of 31 and 33 weeks, respectively. There are some exceptional projects for policies CADB and CBDA where the assignment 2211 shows a good performance; again, we shall discuss that in the next subsection.

Finally, policies which assign the two front-end components and the difficult application component D to the same team also lead to a long project completion time. The assignment 1121 occurs in about 80 percent of the projects for the policies ACBD and BCAD with an average completion time of 30 weeks. The assignment 2212, which occurs in about 15 percent of the projects for the policies DBAC and DABC, causes an extremely long completion time of almost 37 weeks.

Table 6 gives a ranking of the possible assignments, ranked according to their performance averaged over all 12,000 simulation runs of this study.

Table 6. Global ranking of assignments.

assign	75 %	90 %	mean
2121	28	32	26.3
1221	29	32	26.8
1112	29	33	27.1
2112	31	35	28.8
2211	32	36	28.8
1212	32	36	29.6
1121	32	37	29.5
1122	33	37	31.7
2221	40	45	36.5

4.3. Context sensitivity

The average project completion time for some task assignment frequently depends on the *project context* in which that assignment arises. For example, in most cases the assignment 2211, which assigns both large application components to team One, yields a long project completion time. Yet, there are two exceptions: if the assignment occurs for the policy CADB or CBDA, the average completion time is as good as with the best balanced assignment, namely, about 26 weeks. The reason is that for the policies CADB and CBDA the assignment 2211 can only occur if team One finishes the large component C faster than team Two is done with its first front-end component (A or B). Such a project context is not very likely to occur, but points to a fast project completion.

Other task assignments exhibit a similar sensitivity of their performance to the project context. Table 7 shows for each task assignment under which list policies the assignment occurs and what its performance is for that policy. The table shows that the assignments 2211, 2121, 1112, 2112, and 1212 are sensitive to the project context. In the remainder of this subsection, we take a closer look at some examples.

The policies ADBC and BDAC almost always lead to the assignment 1112, but do not exhibit as good a performance as, say, ADCB does with that assignment. The obvious difference is that for ADBC team One works on the large component C *after* both small front-end components. This holds also for BDAC, but not for ADCB and the other policies which lead to assignment 1112. In fact, assignment 1112 can occur for policies such as ADCB only if team One finishes component C and one front-end component more quickly than team Two finishes component D. This setting points to a fast project completion. For ADBC and BDAC, component C is assigned to team One in any case.

Assignment 2112 shows a much better project completion time for the policies BDCA and BADC than for BACD

Table 7. Context sensitivity of assignments.

assign	policy	75 %	90 %	mean
2121	BCDA	28	31	25.7
	DCAB	28	31	26.3
	DCBA	28	32	25.9
	DACB	28	32	26.1
	BADC	29	33	27.2
	BACD	30	37	27.5
1221	ACDB	28	31	26.2
	DCAB	28	32	26.6
	DCBA	29	33	27.0
	DBCA	29	33	27.1
	ABDC	29	33	27.1
1112	ADCB	27	31	25.6
	BDCA	28	32	26.6
	CDAB	29	33	26.5
	CDBA	29	33	26.8
	BDAC	30	34	27.9
	ADBC	30	35	28.0
2112	BDCA	29	33	27.3
	BADC	30	33	27.2
	CDBA	30	33	27.9
	CABD	31	34	28.6
	BACD	31	35	29.5
	CADB	33	37	29.8
2211	CADB	28	33	26.1
	CBDA	28	34	26.2
	DBAC	31	35	28.0
	DABC	32	36	28.5
	CABD	34	39	30.6
	CBAD	34	40	30.3
1212	CDAB	30	33	28.5
	ADCB	31	34	28.4
	CBAD	31	39	30.7
	ABCD	32	37	29.9
	CBDA	33	38	30.3
1121	BCAD	32	37	30.0
	ACBD	33	38	30.2
1122	ACBD	33	35	31.3
	BCAD	34	39	32.7
2221	DBAC	39	44	36.7
	DABC	40	45	36.7

or CADB. For BACD and CADB, assignment 2112 occurs in about 80 percent of the projects. For BDCA, there is a more than 50 percent chance for the alternative assignment 1112 to occur, which in this context leads to a short average completion time of the project. The assignments 2112 and 1112 are in a race condition for policy BDCA, which might explain why the performance of 2112 is closer to the performance of 1112 for policy BDCA than to the performance of 2112 for policy BACD. The same holds for policy BADC, where the alternative assignment 2121 is highly likely and also has a short average completion time.

Some questions remain open. For example, it is not clear yet why assignment 2211 performs better for the policies DABC and DBAC than for CABD and CBAD. Similarly, it is not clear why the good balanced assignment 2121 performs not as well with the policies BADC and BACD as it does with other policies such as DCBA. In order to understand such observations and to confirm some of our findings in this subsection, we need a more detailed analysis of the simulated projects, including an analysis of the net development times and change propagation. This is work in progress.

4.4. Previous observations

We can use the analysis of the task assignments which actually occur for the list policies to explain the observations made in subsection 3.5:

1. The policies which assign the large components first all perform well, but for various reasons. Policies DCAB and DCBA lead to good balanced assignments where team One works on component D. Policies CDAB and CDBA lead to the assignment 1112 or to a balanced assignment where team Two works on component D; both assignments show a good performance in this context. Policies ABDC and BADC lead to the same favorable balanced assignment as DCAB and DCBA.
2. Policies DABC and DBAC take much longer to complete than the other list policies which assign component D first, because DABC and DBAC assign both large components to the same team, or, the large component C and both front-end components to the slower team Two.
3. For various reasons, the list policies where team One starts with component C perform worse than their counterparts where team One starts with component D. The policies CABD, CBAD, DABC, and DBAC in most cases lead to assignment 2211, which shows a better performance if component D is worked on first instead of component C (although it is not yet clear, why). Policies CADB, CBDA, DACB, and DBCA lead to balanced assignments, but DACB and DBCA assign the

large component D to the faster team One. Policies DCAB and DCBA lead to balanced assignments with component D assigned to team One. On the other hand, policies CDAB and CDBA in about half of the projects lead to the assignment 1112, which has a good performance in this context; but for the other projects, policies CDAB and CDBA lead to a less favorable balanced assignment where component D is assigned to the slower team Two.

4. The list policies where team Two starts with the largest component D can all lead to the assignment 1112. For ADBC and BDAC, this assignment occurs in almost all projects and requires on average about 28 weeks before the project is finished. For ADCB, BDCA, CDAB, and CDBA, the assignment 1112 occurs in half of the projects, now requiring only about 26 weeks of development time. This advantage is outweighted though by the other half of the projects which require 27 or 28 weeks to finish, because they lead to a balanced assignment where component D is assigned to team Two.
5. The policies which assign the largest component D last are not a good choice. Policies BACD and ABCD lead to a balanced assignment, but with team Two working on the large application component D. Policies ACBD and BCAD assign component D and both front-end components to the same team. Policies CABD and CBAD assign both application components to the same team, which yields a weak performance in this context.
6. The policies which assign the front-end components second and third are a bad choice. Policies CABD, CBAD, DABC, and DBAC in most projects lead to assignment 2211, which shows a weak performance in those project contexts.

5. Conclusions

For a list policy, the scheduling actions are based on the given priority list. In addition, a list policy uses data about which components are completed and which previously completed components must be reworked. No other information is used. One might ask whether it is possible to achieve a better average project completion time when replacing the list policies by more *dynamic* scheduling strategies. A dynamic strategy would base its scheduling decision at the end of a phase on the full data about the current state of the project, including such data as the current net development times. In addition, the project input data could be utilized, such as data about the coupling between the components or the expected net development times computed from the base probabilities.

In fact, several results of this paper provide clear evidence that strategies which are more adaptive than list policies will yield improved schedules:

- The performance of a task assignment often depends on the project context in which the assignment occurs.
- For a number of list policies which can lead to different task assignments, the performance of the policy varies considerably with the actual assignment. Examples are the list policies ADCB, CADB, CBDA, CDAB, DABC, and DBAC.
- For the sample project, there are three completely different task assignments which show the best performance, namely, 2121, 1112, and 2211. The assignments 1112 and 2211 give a good performance only in a certain project context.

Our scheduling model (as well as our simulation model) is not limited to list policies; any scheduling action can be chosen based on the current project state. Thus, it is promising to apply dynamic schedule optimization techniques to our software project scheduling model, as we have proposed earlier [7]. This is future work.

References

[1] Abdel-Hamid, Madnick: *Software Project Dynamics*. Prentice Hall, 1991

[2] Bertsekas: *Dynamic Programming and Optimal Control*. Athena Scientific, 1995

[3] EXTEND, `http://www.imaginethatinc.com/`

[4] Madachy: "System Dynamics Modeling of an Inspection-Based Process", Proceedings of the International Conference on Software Engineering ICSE 18 (1996) 376-386

[5] Möhring: "Scheduling under Uncertainty: Optimizing Against a Randomizing Adversary", Proceedings 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, Springer LNCS 1913 (2000) 15-26

[6] Neumann: "Scheduling of Projects with Stochastic Evolution Structure", see [10] 309-332

[7] Padberg: "Scheduling Software Projects to Minimize the Development Time and Cost with a Given Staff", Proceedings of the Asia-Pacific Software Engineering Conference APSEC 8 (2001) 187–194

[8] Padberg: "A Stochastic Scheduling Model for Software Projects", Dagstuhl Seminar on Scheduling in Computer and Manufacturing Systems, June 2002, Dagstuhl Report No. 343

[9] Tvedt, Collofello: "Evaluating the Effectiveness of Process Improvements on Software Development Cycle Time via System Dynamics Modeling", Proceedings of the International Computer Software and Applications Conference COMPSAC 19 (1995) 318-325

[10] Weglarz: *Project Scheduling. Recent Models, Algorithms, and Applications*. Kluwer, 1999